Date=29/09/2020
Lecture By=Arkesh Jaiswal
Subject ⇒ Preprocessors

IN PREVIOUS LECTURE **(QUICK RECAP) Date-28/09/2020**	**In Today's Lecture (Overview)**
CSS Flexbox Parent Element (Container) The flex-wrap Property The align-items Property The align-content Property Questions for Self practice / CC for the day	Preprocessors SASS/SCSS Variables Nesting Mixins Operators Questions for the self Practice / CC for the Day

Preprocessors

In computer science, a **preprocessor** is a program that processes its input data to produce output that is used as input to another program. The output is said to be a **preprocessed** form of the input data, which is often used by some subsequent programs like compilers.

SASS/SCSS

Sass (which stands for 'Syntactically awesome style sheets) is an extension of CSS that enables you to use things like variables, nested rules, inline imports and more. It also helps to keep things organised and allows you to create style sheets faster.

Preprocessing

CSS on its own can be fun, but stylesheets are getting larger, more complex, and harder to maintain. This is where a preprocessor can help. Sass lets you use features that don't exist in CSS yet like variables, nesting, mixins, inheritance and other nifty goodies that make writing CSS fun again.

Once you start tinkering with Sass, it will take your preprocessed Sass file and save it as a normal CSS file that you can use in your website.

The most direct way to make this happen is in your terminal. Once Sass is installed, you can compile your Sass to CSS using the `sass` command. You'll need to tell Sass which file to build from, and where to output CSS to. For example, running `sass input.scss output.css` from your terminal would take a single Sass file, `input.scss`, and compile that file to `output.css`.

Variables

Think of variables as a way to store information that you want to reuse throughout your stylesheet. You can store things like colors, font stacks, or any CSS value you think you'll want to reuse. Sass uses the `$` symbol to make something a variable. Here's an example:

SCSS	SASS	CSS

`$font-stack: Helvetica, sans-serif;`
`$primary-color: #333;`
 	
`body {`
`  font: 100% $font-stack;`
`  color: $primary-color;`
`}`
	`$font-stack: Helvetica, sans-serif`
	`$primary-color: #333`
	
	`body`
	`  font: 100% $font-stack`
	`  color: $primary-color`
		`body {`
		`  font: 100% Helvetica, sans-serif;`
		`  color: #333;`
		`}`

When the Sass is processed, it takes the variables we define for the $font-stack and $primary-color and outputs normal CSS with our variable values placed in the CSS. This can be extremely powerful when working with brand colors and keeping them consistent throughout the site.

Nesting

When writing HTML you've probably noticed that it has a clear nested and visual hierarchy. CSS, on the other hand, doesn't.
Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML. Be aware that overly nested rules will result in overqualified CSS that could prove hard to maintain and is generally considered bad practice.
With that in mind, here's an example of some typical styles for a site's navigation:

SASS	SCSS	CSS

nav
  ul
    margin: 0
    padding: 0
    list-style: none
 	
  li
    display:
inline-block
 	
  a
    display: block
    padding: 6px 12px
    text-decoration:
none
	nav {
	  ul {
	    margin: 0;
	    padding: 0;
	    list-style: none;
	  }
	
	  li { display:
	inline-block; }
	
	  a {
	    display: block;
	    padding: 6px 12px;
	    text-decoration:
	none;
	  }
	}
		nav ul {
		  margin: 0;
		  padding: 0;
		  list-style: none;
		}
		nav li {
		  display:
		inline-block;
		}
		nav a {
		  display: block;
		  padding: 6px 12px;
		  text-decoration:
		none;
		}

You'll notice that the ul, li, and a selectors are nested inside the nav selector. This is a great way to organize your CSS and make it more readable.

Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible. A good use of a mixin is for vendor prefixes. Here's an example for transform.

SASS	SCSS	CSS
`=transform($property)` `-webkit-transform: $property` `-ms-transform: $property` `transform: $property` `.box` `+transform(rotate(30deg))`	`@mixin transform($property) {` `-webkit-transform: $property;` `-ms-transform: $property;` `transform: $property;` `}` `.box { @include transform(rotate(30deg)); }`	`.box {` `-webkit-transform: rotate(30deg);` `-ms-transform: rotate(30deg);` `transform: rotate(30deg);` `}`

Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like `+`, `-`, `*`, `/`, and `%`. In our example we're going to do some simple math to calculate widths for an `aside` & `article`.

SASS	SCSS	CSS
`.container` `width: 100%` `article[role="main"]`	`.container {` `width: 100%;` `}` `article[role="main"] {`	`.container {` `width: 100%;` `}` `article[role="main"] {`

`float: left`
`width: 600px / 960px * 100%`
 	
`aside[role="complementary"`
`float: right`
`width: 300px / 960px * 100%`
	`float: left;`
	`width: 600px / 960px * 100%;`
	`}`
	
	`aside[role="complementary"] {`
	`float: right;`
	`width: 300px / 960px * 100%;`
	`}`
		`float: left;`
		`width: 62.5%;`
		`}`
		
		`aside[role="complementary"] {`
		`float: right;`
		`width: 31.25%;`
		`}`

Questions for the self Practice / CC for the Day

Design a ecommerce website like flipkart/myntra with the following components:
- Header with logo and important links
- Sidebar with multiple filters
- Main content area with product grid layout
- Footer with set navigation